VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Kryptografie napříč programovacími jazyky

Cryptography across Programming Languages

2013 **Pavel Michálek**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Michálek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Kryptografie napříč programovacími jazyky
Cryptography across Programming Languages

Zásady pro vypracování:

Cílem práce je experimentálně prozkoumat a zhodnotit možnosti kryptografie ve vybraných programovacích jazycích (Java, C#, Python, PHP). Zaměřte se na šifry 3DES, AES (Rijndael) a RSA.

1. Popište možnosti kryptografie ve vybraných jazycích (integrované funkce, popř. nejrozšířenější přídavné knihovny).
2. V jednotlivých jazycích vytvořte programy používající vybrané šifry.
3. Experimentálně prověřte možnosti vzájemné spolupráce. Výsledky dokumentujte a sestavte doporučení, podle kterých lze případné problémy spolupráce vyřešit.

Seznam doporučené odborné literatury:

[1] Menezes, Oorschot, Vanstone, Scott. Handbook of Applied Cryptography. ISBN: 0-8493-8523-7 <http://cacr.uwaterloo.ca/hac/>
[2] National Institute of Standards and Technology. Data Encryption Standard (DES).<http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>
[3] National Institute of Standards and Technology. Advanced Encryption Standard (AES). <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
[4] PHP Manual, Cryptography Extensions. <http://php.net/manual/en/refs.crypto.php>
[5] The Ruby Toolbox, Encryption. <https://www.ruby-toolbox.com/categories/encryption>
[6] PyCrypto - The Python Cryptography Toolkit. <https://www.dlitz.net/software/pycrypto/>
[7] Delfs, Hans, and Helmut Knebl. Introduction to cryptography: principles and applications. Springer, 2007
[8] Bulant, Michal. "Kryptografie a její aplikace"

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jakub Macek**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

„Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal."

Datum: 7.5.2013

Podpis: 

Poděkování

Rád bych poděkoval Ing. Jakubovi Mackovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

Abstrakt

Cílem práce je experimentálně prozkoumat a zhodnotit možnosti kryptografie v programovacích jazycích Java, C#, Python a PHP se zaměřením na šifry 3DES, AES a RSA. V těchto vybraných jazycích potom popsat možnosti kryptografie, jejich integrované funkce i nejrozšířenější externí knihovny. Konečným cílem této práce je prověřit možnosti vzájemné spolupráce jazyků.

První část práce se zabývá popisem šifer, jejich využití a výčtem silných a slabých stránek. Obsahem druhé části je šifrování v jednotlivých jazycích, ukázky zdrojových kódů, rozdíly mezi jazyky a srovnání integrovaných funkcí a externích knihoven. Ve třetí části čtenář nalezne řešení problémů, které provázejí spolupráci programovacích jazyků v kryptografii, zdrojové kódy a doporučení autora.

Klíčová slova

kryptografie, programování, spolupráce, součinnost, Java, C#, PHP, Python, 3DES, AES, RSA

Abstract

The aim is to experimentally explore and assess options of cryptography in programming languages Java, C #, Python, and PHP, with focus on ciphers 3DES, AES and RSA. In these selected languages describe the possibilities of cryptography, integrated functions and external libraries. The ultimate goal of this work is to examine the possibilities of mutual cooperation between languages.

The first part deals with the description of ciphers, their use and a list of strengths and weaknesses. The second part of is the encryption in different languages, examples of source code, differences between languages and comparison of integrated functions and external libraries. In the third section the reader will find solutions to the problems that accompany cooperation programming languages in cryptography, source codes and recommendations of the author.

Key words

Cryptography, programming, cooperation, interoperability, Java, C#, PHP, Python, 3DES, AES, RSA

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
AES	Advanced Encryption standard	Pokročilý šifrovací standard
CBC	Cipher Block Chaining	Řetězení bloků šifry
CFB	Cipher Feedback	Zpětná vazba šifry
DES	Data Encryption Standard	Datový šifrovací standard
ECB	Electronic Codebook	Elektronická kódová kniha
EDE	Encrypt Decrypt Encrypt	Šifrovat dešifrovat šifrovat
EEE	Encrypt Encrypt Encrypt	Šifrovat šifrovat šifrovat
IV	Initialization vector	Inicializační vektor
MIT	Massachussets Institute of Technology	
NIST	National Institute of Standards and Technology	Národní institut standardů a technologie
OAEP	Optimal Asymmetric Encryption Padding	Optimální asymetrická šifrovací výplň
OFB	Output Feedback	Výstup zpětné vazby
PKCS	Public-Key Cryptography Standards	Standard pro šifrování veřejným klíčem
PKCS#5	Password-based Encryption Standard	Standard pro hesla
PKCS#7	Cryptographic Message Syntax Standard	Standard pro syntaxi šifrovacích zpráv
WiFi	Wireless Fidelity	Bezdrátová síť
WPA2	WiFi Protected Access	Zabezpečení přístupu k WiFi
md5	Message-Digest algorithm	Algoritmus otisku zprávy

Obsah

1 Úvod

Dnešní rozsáhlé využívání výpočetní a komunikační techniky vede ve svých důsledcích ke stále se zvětšujícímu objemu zpracováváných dat a ukládaných informací. Vzrůstá zranitelnost těchto dat a není žádnou novinkou, že potenciální narušitelé dokáží tato data získat a přivodit nám ekonomickou ztrátu. Proto vzniká daleko větší potřeba šifrované komunikace a ukládaní dat, a to v různých, běžně používaných, programovacích jazycích. Popíšeme si tedy hlavní symetrické a asymetrické šifry a ukážeme si příklady těchto šifer v jednotlivých programovacích jazycích. V první části se podíváme na šifry a teoreticky si je rozebereme, ukážeme si v čem tkví jejich silná stránka a naopak v čem upadají. V další části si tyto šifry implementujeme do programovacích jazyků, dozvíme se kdy musíme šáhnout po externí knihovně a kdy si vystačíme s integrovaným algoritmem. Ve třetí a poslední části se budeme zabývat spolupráci jednotlivých jazyků a co všechno musíme znát, aby se taková spolupráce podařila.

2 Kryptografie

Nejprve bychom si měli říct co to kryptografie je. Slovo Kryptografie pochází z řečtiny – kryptós (skrytý) a gráphein (psát). Kryptografie, neboli šifrování, je nauka o metodách utajování smyslu zpráv převodem do podoby, která je čitelná jen se speciální znalostí. Jednodušeji řečeno jde o znehodnocování textu takovým způsobem, aby byl zasvěcený člověk schopen si tento text navrátit do původní podoby. Jako doložené počátky šifrování je možno brát v úvahu již dobu starých Egypťanů, kteří používaly pro zapisování osobních a citlivých dat speciálně upravené hieroglyfy, v nichž se vyznala jen úzká skupina lidí, která s nimi byla seznámena. Pokud bychom šli do teoretičtější roviny, je možno šifrováním nazvat i běžné hieroglyfy, vzhledem k tomu, že většina z nás hieroglyfy neumí číst, natož pak psát.

Než začneme s popisem jednotlivých šifer tak si vysvětlíme základní pojmy které se s těmito šiframi pojí.

2.1 Šifra nebo šifrování

Je to kryptografický algoritmus, který převádí původní prostý text na znehodnocenou (nečitelnou) podobu, neboli šifrovaný text.

2.2 Klíč

Tajná informace, která nám umožní šifrovanou zprávu dešifrovat. Bez klíče zašifrovanou zprávu nepřečteme.

2.3 Symetrická šifra

Šifrovací algoritmus, který používá pro šifrování i dešifrování stejný klíč. Výhodou takové šifry je nízká výpočetní náročnost, proto je vhodná k šifrování velkého objemu dat. Nespornou nevýhodou je naopak to, že odesílatel i příjemce musí sdílet stejný klíč, který si musí předat zabezpečeným způsobem.

2.4 Asymetrická šifra

Šifrovací algoritmus, který na šifrování a dešifrování používá jiné klíče – veřejný a soukromý. Tato technologie je založena na jednocestných funkcích, což jsou operace, které lze snadno provést pouze v jednom směru - ze vstupu lze snadno spočítat výstup, z výstupu však je velmi obtížné nalézt vstup. Příkladem je násobení. Je snadné vynásobit mezi sebou dvě velká čísla, ale rozklad součinu na činitele (faktorizace) je velmi obtížný.

Obecně se v této spojitosti používají postavy Alice a Bob, vysvětlíme si to tedy na nich. Řekněme, že Bob chce poslat zprávu Alici, která je tajná. Alice si tedy vygeneruje dvojici klíčů – veřejný a soukromý. Veřejný klíč pošle Bobovi, nebo jej rovnou zveřejní, nezáleží na tom, kdo o veřejném klíči ví. Bob zašifruje zprávu tímto veřejným klíčem a takto šifrovanou zprávu pošle Alici. Ta ji dešifruje klíčem soukromým a přečte. Jak z příkladu vidíme, zpráva se zašifruje veřejným klíčem, ale dešifrovat tím samým klíčem nejde, na dešifrování se musí použít klíč soukromý, který má jen Alice. Velkou výhodou této metody je skutečnost, že odpadá riziko při posílání klíče. Nevýhodou je potom vysoká výpočetní náročnost, takže se příliš nehodí na šifrování objemných dat. Obvykle se

asymetrickými šiframi šifruje jen jeden blok. Pokud je zvolena délka klíče 1024 bitů, je délka bloku rovna také 1024 bitů, tedy 128 bytů. Musíme však odečíst 11 bytů, které jsou obětovány paritě. Pro 1024 bitů dlouhý klíč tedy dostáváme blok o délce 117 bytů.

Pokud bychom se podívali na výhody a nevýhody symetrických a asymetrických šifer tak dojdeme k závěru, že při spojení těchto dvou metod dostaneme opravdu silný kryptografický nástroj, který přejímá výhody obou metod. Objemnou zprávu zašifrujeme symetrickou šifrou a její klíč asymetrickou, klíč následně pošleme.

2.5 Bloková šifra

Jedná se o typ symetrické šifry, která je rozdělena do bloků pevně stanovené délky, např. 128 bitů. Pokud jsou šifrovaná data delší, tak se rozdělí do více bloků, pokud jsou kratší, což se týká většinou posledního bloku, tak je potřeba blok doplnit na odpovídající délku. K tomu slouží výplň (anglicky padding). Existuje několik algoritmů které nám tuto výplň zajistí, od jednoduchého způsobu doplnění nulami až po složitější. Nejčastější výplní je podle standardu PKCS#7, se kterým se setkáme v dalších částech této práce. U blokových šifer rozlišujeme taky různé druhy režimu provozu, které určují jakým způsobem algoritmus pracuje.

2.5.1 Operační mody blokových šifer:

2.5.1.1 Mod ECB

Nejjednodušším modem je ECB, který se ale nedoporučuje. Výsledkem tohoto modu je, že všechny bloky budou zašifrovány stejně. Protože bloky jsou poměrně malé, tak hrozí nebezpečí, že v zašifrované zprávě budou patrná stejná schémata, jako v původní zprávě. Další nebezpečí je vyměňování nebo vkládání bloků šifry, jelikož bloky na sobě nejsou nijak závislé.

2.5.1.2 Mod CBC

Ten funguje tak, že před zašifrováním se odpovídající blok otevřeného textu XORuje předchozím blokem zašifrovaného textu. To znamená, že jednotlivé bloky jsou na sobě závislé a pro dešifrování jednoho bloku se musí dešifrovat i bloky předchozí. Čím ale XORovat první blok? Jeden blok se vygeneruje náhodně a přidá se k zašifrovaným datům jako nultý blok. Tomuto nultému bloku se říká inicializační vektor. Tento vektor se používá jenom k dešifrování prvního bloku a potom se zahodí. Je tedy patrné, že inicializační vektor potřebujeme pouze u módu CBC, mód ECB nic takového nepotřebuje. CBC je nejpoužívanějším modem blokových šifer.

2.5.1.3 Metoda solení

U modu CBC je možno využít metodu solení IV. V praxi se jedná o to, že komunikujícímu protějšku se předává hodnota IV, ale k šifrování se použije „osolená“ IV. Tato hodnota se na obou stranách vypočítá nějakým předem definovaným způsobem. Např. se může jednat o hašování IV a klíče. Výhodou této metody je, že skutečně použítá IV se nikde na komunikačním kanále neobjevuje.

2.5.1.4 Mody CFB a OFB

Tyto operační mody převádí blokovou šifru na proudovou. Používají náhodnou inicializační hodnotu IV k nastavení odpovídajícího konečného automatu do náhodné polohy. Tento automat pak produkuje posloupnost hesla, které se jako u proudových šifer XORuje na otevřený text. První blok hesla se získá zašifrováním IV. Konečný automat pracuje tak, že vzniklé heslo (v modu OFB) nebo

vzniklý šifrový text (v modu CFB) jsou vedeny na vstup blokové šifry a jejich zašifrováním je produkován následující blok hesla. OFB má vlastnost čisté (synchronní) proudové šifry, neboť heslo je generováno zcela autonomně bez vlivu otevřeného a šifrového textu. CFB je kombinací vlastností CBC a proudové šifry.

2.6 Proudová šifra

Symetrická šifra, u které se vstupní datový tok zkombinuje s pseudonáhodným proudem bitů vytvořeným z šifrovacího klíče a šifrovacího algoritmu. Výsledkem je zašifrovaný proud dat. Výhodou těchto šifer je jejich rychlost a menší náročnost na hardware, naopak nevýhodou může být větší náchylonost k útokům, pokud jsou nevhodně implementovány.

3 Triple DES

Je to symetrická bloková šifra, která je založena na Data Encryption Standard. Algoritmus spočívá v aplikování šifry DES třikrát na každý blok. Je to poměrně snadná metoda jak zvýšit bezpečnost šifry proti útoku hrubou silou bez toho, aby se musel vyvíjet zcela nový algoritmus. Místo jednoho klíče je zde balík tří klíčů, velký 64, 128 nebo 192 bytů.

Rozlišujeme tři verze klíčů a 2 módy algoritmu. První verze spočívá v tom, že jsou všechny 3 klíče odlišné. Druhá verze je taková, že první a třetí klíče jsou stejné, druhý klíč je odlišný. Třetí verze je varianta, kde jsou všechny klíče stejné. Varianty algoritmu jsou EEE a EDE. Mód EEE spočívá v tom, že se celý blok třikrát zašifruje, pokaždé jedním z klíčů. Mód EDE probíhá tak, že se původní zpráva zašifruje prvním klíčem, dešifruje druhým klíčem a následně zašifruje třetím klíčem. Dešifrování probíhá přesně naopak, tzn. dešifrování šifrované zprávy třetím klíčem, zašifrování druhým klíčem a dešifrování prvním klíčem. Tento postup zvyšuje bezpečnost algoritmu.

3.1 Bezpečnost

Nejbezpečnější je používat mód EDE a první variantu klíče v kombinaci s módem CBC. Třetí verze klíče s modem EDE je ekvivalent k šifře DES s delším klíčem, protože se první a druhý krok algoritmu navzájem anulují. Tato metoda není doporučována.

Triple DES je pomalejší než DES, ale nabízí daleko větší bezpečnost proti útokům hrubou silou. Používá se např. v systémech elektronického bankovnictví nebo v programech firmy Microsoft na ochranu uživatelských dat. Společně se šifrou AES je platným oficiálním standardem nahrazujícím DES.

4 AES (Rijndael)

V roce 1997 vyhlásil Americký úřad pro standardizaci veřejnou soutěž o novou šifru, která by nahradila již zastaralou a ne příliš bezpečnou šifru DES. Název této nové šifry byl AES. Do soutěže se přihlásilo 15 autorů a v roce 2001 NIST schválil Rijndael jako standard pro šifrování amerických vládních. Název Rijndael je složen z počátečních písmen autorů šifry, pánů Rijmen a Daemen.

AES je symetrická šifra, její klíč může být dlouhý 128, 196 nebo 156 bitů. Hlavní výhoda je rychlost algoritmu, což nám umožňuje šifrovat velké množství dat. Algoritmus zašifrování i dešifrování se dá výhodně programovat na různých typech procesorů, má malé nároky na operační paměť i velikost kódu a je vhodný i pro paralelní zpracování. Zajímavostí je, že příprava klíčů pro dešifrování a šifrování je odlišná, proto je dešifrování náročnější. Šifra se používá např. pro bezdrátové Wifi sítě v rámci zabezpečení WPA2.

4.1 Bezpečnost

AES je považována ze velmi bezpečnou, útok hrubou silou by vyžadoval 2^{200} operací, což je obrovské číslo. Tato šifra je také odolná proti známým útokům a metodám lineární a diferenciální kryptoanalýzy. Avšak existuje několik úspěšných útoků na implementaci na daném systému, což by mohlo vést k úniku dat. Tento druh útoku se nazývá útok postranními kanály. Neútočí na algoritmus šifry, ale snaží se data získat jinou cestou, pokud by se jednalo o stolní počítač tak se útok může zaměřit na časovou náročnost, teplotu, vyrovnávací paměť apod. V roce 2005 oznámil D. J. Bernstein prolomení uživatelského serveru pomocí cache-timing útoku. (BERNSTEIN, Daniel. *Cache-timing attacks on AES* [online]. 2005 [cit. 7.5.2013]).

5 RSA

Algoritmus RSA byl prvním šifrovým systémem vytvořeným po zveřejnění myšlenky asymetrické kryptografie. Řadí se mezi nejpoužívanější algoritmy v oblasti bezpečných komunikací. Název je složen z počátečních písmen autorů Rivesta, Shamira a Adlemana, kteří algoritmus vypracovali v roce 1977 na prestižním Massachussets Institute of Technology. Jedná se o první algoritmus, který je vhodný k podepisování i šifrování.. Je postaven na předpokladu, že rozložit velké číslo na součin prvočísel (faktorizace) je velmi obtížná úloha. Pokud máme číslo n=p*q, pak je v rozumném čase téměř nemožné zjistit činitele p a q. Není znám žádný algoritmus, který by pracoval v polynomiálním čase vůči velikosti binárního zápisu čísla n. Oproti tomu je vynásobení dvou velkých čísel elementární úloha.

Algoritmus RSA se používá s drobnými úpravami dodnes a snad opravdu všude – u mobilních telefonů, bankomatů, elektronických podpisů atd. Vzhledem k tomu, že se jedná o šifru asymetrickou, tudíž nevhodnou pro šifrování objemných dat, se RSA používá hlavně pro šifrování klíčů symetrických šifer. Častěji se toto využití asymetrických šifer nazývá výměnou klíčů (příp. nastavením klíčů, dohodou klíčů). Druhé použití asymetrických šifer je při realizaci digitálního podpisu. Asymetrická šifra v tomto případě opět nezpracovává vlastní podepisovaný soubor dat, ale pouze jeho hašovací hodnotu.

5.1 Bezpečnost RSA

Algoritmus je považován za bezpečný, pokud se použije správná délka klíče. V dnešní době se doporučuje používat nejméně 2048 bitů. Aby se zvýšila bezpečnost, tak se začalo využívat schéma doplnění. Jde o použití strukturované náhodné posunutí hodnoty *m* (původní zpráva) než jak je zašifrována. Toto posunutí zaručuje, že *m* nebude spadat do rozsahu nebezpečných původních textů, a že daná zpráva po posunutí zašifruje do různých možných zašifrovaných textů. K tomu to účelu se používají standrardy jako PKCS#1 nebo OAEP.

5.2 Útoky

Protože je RSA deterministický šifrovací algoritmus (tzn. že nemá žádnou náhodnou část), tak může útočník zašifrovat předpokládaný text veřejným klíčem a porovnat jej se zašifrovaným textem. Pokud útočník není schopen rozlišit od sebe dva zašifrované texty i když zná původní text, pak je šifra sémanticky bezpečná. K tomu aby toto splňovala musí obsahovat náhodné doplnění.

5.3 Digitální podpis

Základním principem takového využití je opačné použití šifry. Pokud chce Alice poslat Bobovi podepsanou zprávu, připojí k ní číslo získané jakoby dešifrováním haše své zprávy pomocí soukromého klíče. Bob poté zpětně zašifruje tento podpis pomocí Alicina veřejného klíče a porovná výsledek s hašem zprávy. Pokud zpráva nebyla změněna, vyjde stejná hodnota, neboť algoritmus je z hlediska šifrování i dešifrování symetrický. A protože jedině Alice zná soukromý klíč, je tím potvrzeno, že zprávu poslala skutečně ona.

6 Triple DES

Nyní se zaměříme na implementování algoritmu Triple DES v programovacích jazycích C#, Java, PHP a Python.

6.1 Triple DES v C#

Prostředí .NET nabízí přímo třídu TripleDESCryptoServiceProvider, což je třída obsahující algoritmy a atributy, které budeme potřebovat při šifrování TripleDES. Nemusíme tedy používat externí knihovny. Po inicializace provideru dostaneme možnost si nastavit mod a výplň. Defaultně jsou nastaveny na mod CBC a výplň podle standardu PKCS#7. Tyto varianty jsou nejbezpečnější a jejich použití se doporučuje, nicméně pokud bychom chtěli, můžeme zvolit jakýkoliv z modů blokových šifer, z výplní potom máme na výběr ze standardů ANSIX923, ISO10123, a z možností Zeros a None (bez výplně). Pokud si sami nenastavíme klíč a inicializační vektor, potom si provider vytvoří svoje. Takto vytvřený klíč má délku 192 bitů a IV má délku 64 bitů.

Zdrojový kód:

Šifrování:

```
// inicializace provideru
TripleDESCryptoServiceProvider tDESalg = new ripleDESCryptoServiceProvider();

// vytvoření streamu pro zálohování dat
MemoryStream mStream = new MemoryStream();

// vytvoření streamu pro šifrování, který využívá stream pro zálohu, klíč a
// IV
CryptoStream cStream = new CryptoStream(mStream, new
TripleDESCryptoServiceProvider().CreateEncryptor(Key, IV),
CryptoStreamMode.Write);

// konvertujeme původní data do pole bytů
byte[] toEncrypt = new ASCIIEncoding().GetBytes(Data);

// zapíšeme do streamu
cStream.Write(toEncrypt, 0, toEncrypt.Length);
cStream.FlushFinalBlock();

// vytvoříme novou proměnnou do které uložíme data, které nám drží stream
// pro zálohu
byte[] ret = mStream.ToArray();

// oba streamy uzavřeme a zašifrovanou zprávu pošleme na výstup
cStream.Close();
mStream.Close();
return ret;
```

Nejprve si vytvoříme stream pro zálohování dat, poté si vytvoříme druhý stream pro šifrování který využívá stream pro zálohování, klíč a IV. Konvertujeme text původních dat do pole bytů a zapíšeme do streamu. Následně zapíšeme data do nové proměnné ze streamu pro zálohování dat, který nám drží data ze stremu pro šifrování. Nakonec oba streamy uzavřeme.

Dešifrování

```
// vytvoření streamu pro zálohování dat
MemoryStream msDecrypt = new MemoryStream(Data);
```

```
// vytvoření streamu pro šifrování, který využívá stream pro zálohu, klíč a
// IV
CryptoStream csDecrypt = new CryptoStream(msDecrypt,
new
TripleDESCryptoServiceProvider().CreateDecryptor(Key, IV),
CryptoStreamMode.Read);

// vytvoření nové proměnné pro šifru
byte[] fromEncrypt = new byte[Data.Length];

// dešifrování a navrácení hodnoty
csDecrypt.Read(fromEncrypt, 0, fromEncrypt.Length);
return new ASCIIEncoding().GetString(fromEncrypt);
```

Z kódu je patrné, že není velký rozdíl mezi šifrováním a dešifrováním. Místo CreateEncryptor použijeme CreateDecryptor a jako Data nám neposlouží originální text, ale již šifrovaná data. Samozřejmě, že na konci by bylo vhodné oba streamy uzavřít. Výhodou programování TripleDES šifry v C# je skutečnost, že nemusíme stahovat žádné externí knihovny, všechny prostředky nám nabízí prostředí .NET. Zároveň i přednastavený mod a výplň na hodnoty, které jsou nejbezpečnější, jsou velmi dobrou variantou. Absence manuálního generování klíčů je také věc, která může být užitečnou.

6.2 Triple DES v Javě

Pro kryptografii v Javě slouží třída Cipher, která se využívá u většiny šifer. Při inicializaci nové instance třídy zadáme jako parametry druh šifry, mod a výplň. Pomocí třídy SecretKey si vytvoříme nový klíč a pomocí třídy IvParameterSpec si definujeme inicializační vektor. Klíč je možné si nechat vygenerovat s použitím třídy KeyGenerator. Tato třída nám vytvoří klíč speciálně pro šifru kterou chceme, stačí při vytvoření instance předat konstruktoru název šifry jako parametr. Další možností by bylo klíč rovnou napsat, nebo si napsat text a na ten použít jednu z hašovacích funkcí, např. SHA1 nebo md5. Tento postup pomocí hašovací funkce si ukážeme. Jediné, na co si musíme dávat pozor, je délka klíče. Pokud zvolíme 24 bytů (což je 192 bitů) tak to je optimální a vychází nám 8 bytů na jeden „podklíč“. Inicializační vektor musí být 8 bytů velký.

Zdrojový kód:

Šifrování

```
// vytvoření instance pro hašování, hašování našeho stringu a tvoření
// klíče, kontrola velikosti klíče
final MessageDigest md = MessageDigest.getInstance("md5");
final byte[] digestOfPassword = md.digest("HG58YZ3CR9".getBytes("utf-8"));
final byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24);
for (int j = 0, k = 16; j < 8;) {
keyBytes[k++] = keyBytes[j++];
}

// vytvoření klíče a předání hodnoty našeho hašovaného stringu, určení typu
// šifry
final SecretKey key = new SecretKeySpec(keyBytes, "DESede");

// vytvoření inicializačního vektoru o velikosti 8 bytů
final IvParameterSpec iv = new IvParameterSpec(new byte[8]);

// vytvoření instance šifry a její inicializace pro šifrování s typem
// šifry, modem a výplní
final Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key, iv);
```

```
// nová proměnné pro šifrování a následné vrácení šifrované zprávy
final byte[] plainTextBytes = message.getBytes("utf-8");
final byte[] cipherText = cipher.doFinal(plainTextBytes);
return cipherText;
```

Dešifrování

```
// vytvoření instance šifry a její inicializace pro dešifrování s typem
// šifry, modem a výplní
final Cipher decipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
decipher.init(Cipher.DECRYPT_MODE, key, iv);

// dešifrování šifrované zprávy a její navrácení
final byte[] plainText = decipher.doFinal(message);
return new String(plainText, "UTF-8");
```

Dešifrování probíhá velmi podobně jako v případě programu v C#. Kód je téměř totožný, až na dva rozdíly. Musíme si vytvořit novou proměnnou a nastavit ji mod jako DECRYPT, abychom mohli dešifrovat.

Název šifry pro Triple DES je DESede nebo TripleDES. V Javě se používá standard PKCS#5 pro výplň, ten je ale téměř totožný jako PKCS#7. Znalost jednotlivých standardů a jejich použití v různých programovacích jazycích bude klíčová v další části práce, kdy se budeme zabývat šifrováním a dešifrováním v různých jazycích.

6.3 Triple DES v PHP

V jazyce PHP se nabízí použít knihovnu pro šifrování zvanou Mcrypt. Tato knihovna podporuje velké množství blokových algoritmů jako je DES, TripleDES nebo Rijndael. Všechno v modech CBC, OFB, CFB, ECB, NOFB nebo stream. Jedinou nevýhodou může být skutečnost, že není integrován žádný ze standardů výplně. Pro jednoduchý algoritmus na vyzkoušení Mcrypt můžeme použít mod ECB. U klíče a IV si nastavíme přes mcrypt potřebnou délku a IV si necháme vygenerovat. Klíč si uděláme sami opět pomocí hašovací funkce md5.

Zdrojový kód:

```
// otevření modulu s předáním parametrů pro název šifry a mod
$td = mcrypt_module_open('tripledes', '', 'ecb', '');

// vytvoření inicializačního vektoru
$iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_DEV_RANDOM);

// vytvoření klíče s přesně definovanou délkou
$ks = mcrypt_enc_get_key_size($td);
$key = substr(md5('very secret key'), 0, $ks);

// inicializace
mcrypt_generic_init($td, $key, $iv);

// vytvoření originální zprávy
$originalText = 'This is very important data';

// zašifrování originální zprávy
$encrypted = mcrypt_generic($td, $originalText);

// deinicializace
mcrypt_generic_deinit($td);
```

```
// inicializace, dešifrování a následná deinicializace a zavření modulu
mcrypt_generic_init($td, $key, $iv);
$decrypted = mdecrypt_generic($td, $encrypted);
mcrypt_generic_deinit($td);
mcrypt_module_close($td);

// zobrazení dat
echo "Original text: ".$originalText;
echo "\n Encrypted: ".$encrypted;
echo "\n Decrypted: ".$decrypted;
```

Při inicializaci si otevíráme modul algoritmu, na konci programu ho musíme zase zavřít. Jako parametry uvádíme název algoritmu (v našem případě „tripledes“), soubor s algoritmem, mod (ecb) a soubor s modem. Pokud cesty k souborům neuvedeme, tak se použijí defaultní.

6.4 Triple DES v Pythonu

V Pythonu máme na výběr z několika balíků. Pravděpodobně nejlepší je PyDES, balík vyloženě pro použití šifry DES a TripleDES. Ten si také podrobně ukážeme. Jeho výhody jsou podpora standardu výplně (PKCS#5), podpora všech modů a podrobná dokumentace na oficiálních stránkách. Naprogramovat tak nějaký vlastní program nepředstavuje žádný problém. Nevýhodou by mohla být menší rychlost. Další skvělou knihovnou je Chillkat Python Encryption Library, která je ale placená. Potom je zde knihovna PyCrypto, která by práci svedla stejně dobře jako PyDES. Další možností je použít knihovnu M2Crypto, která je sice rychlejší než PyDES, ale neobejde se bez nevýhod. Tou hlavní je skutečnost, dokumentace je velmi strohá a ne příliš podrobná.

Zdrojový kód:

```
# importování knihovny pyDdes
from pyDes import *

# vytvoření originálního textu
data = "Test of TripleDES cipher"

# inicializace a následné zašifrování
e = triple_des('a 16 or 24 byte password').encrypt(data, padmode=2)

# inicializace a následné dešifrování
d = triple_des('a 16 or 24 byte password').decrypt(e, padmode=2)

# tisk dat na obrazovku
print "Original data: %r" % data
print "Encrypted: %r" % e
print "Decrypted: %r" % d
input()
```

Jak vidíme, zdrojový kód v Pythonu je podstatně kratší než např. v C# Je to dáno tím, že funkce jsou zapsány v knihovně, kterou si importujeme, náš kód obsahuje pouze volání funkcí. Klíč musí být dlouhý 16 nebo 24 bytů (tento požadavek jsme si určili jako klíč samotný). Inicializační vektor zde nepoužíváme, protože jsme nezadali mod a defaultní nastavení je ECB. Při inicializaci šifry se předávají 4 parametry, jsou to klíč(který zde máme), mod, IV, pad (což je volitelný argument, který nastavuje charakter výplně při defaultním modu výplně) a padmode (mod výplně).

7 AES

Šifrování algoritmem Rijndael v jazycích C#, Java, PHP a Python

7.1 AES v C#

Přístup k algoritmu Rijndael nám zajistí třída RijndaelManaged, opět stejně jako u TripleDES není třeba žádného hledání, vše nám poskytuje .NET. Defaultní hodnoty pro mod a výplň jsou stejné jako u TripleDES. Funkci, kterou si ukážeme předáváme jako parametr původní text, heslo (ze kterého bude klíč), řetězec jako sůl, hašovací algoritmus (např. SHA1), iteraci klíče, inicializační vektor a velikost klíče v bitech. Klíč si tedy vyrobíme sami s použitím hašování a soli. Pokud bychom chtěli jednodušší algoritmus tak nám postačí původní zpráva, klíč a IV v modu CBC a původní zpráva a klíč v modu ECB.

Zdrojový kód:

Šifrování

```
// hlavička funkce
public static string Encrypt(string PlainText, string Password, string Salt,
string HashAlgorithm, int PasswordIterations, string InitialVector, int
KeySize){

// kontrola jestli se nejedná o prázdný původní text
if (string.IsNullOrEmpty(PlainText))
return "";

// převod hodnot ve formě řetězce do pole bytů
byte[] InitialVectorBytes = Encoding.ASCII.GetBytes(InitialVector);
byte[] SaltValueBytes = Encoding.ASCII.GetBytes(Salt);
byte[] PlainTextBytes = Encoding.UTF8.GetBytes(PlainText);

// hašování za použití hesla, soli, určení hašovacího algoritmu a iterace
PasswordDeriveBytes DerivedPassword = new PasswordDeriveBytes(Password,
SaltValueBytes, HashAlgorithm, PasswordIterations);

// vytvoření klíče o velikosti v bytech
byte[] KeyBytes = DerivedPassword.GetBytes(KeySize / 8);

// vytvoření instance
RijndaelManaged SymmetricKey = new RijndaelManaged();

// určení modu šifry, toto je volitelné, jelikož defaultní nastavení je CBC
SymmetricKey.Mode = CipherMode.CBC;

// inicializace proměnné pro šifru
byte[] CipherTextBytes = null;

// vytvoření Encryptoru a paměťového a šifrovacího streamu
using (ICryptoTransform Encryptor = SymmetricKey.CreateEncryptor(KeyBytes,
InitialVectorBytes)){
using (MemoryStream MemStream = new MemoryStream()){

// zašifrování a uložení do streamu a následné navrácení šifrovaného textu ve
// formátu Base64
using (CryptoStream CryptoStream = new CryptoStream(MemStream, Encryptor,
CryptoStreamMode.Write)){
CryptoStream.Write(PlainTextBytes, 0, PlainTextBytes.Length);
CryptoStream.FlushFinalBlock();
CipherTextBytes = MemStream.ToArray();
```

```
                    MemStream.Close();
                    CryptoStream.Close();
                }
            }
        }
        SymmetricKey.Clear();
        return Convert.ToBase64String(CipherTextBytes);
```

Dešifrování

Dešifrování je velice podobné šifrování, až na pár rozdílů, které si ukážeme.

```
// konvertování z Base64 do pole bytů
byte[] CipherTextBytes = Convert.FromBase64String(CipherText);

// vytvoření Decryptoru, streamů a následné dešifrování a vrácení hodnoty
using (ICryptoTransform Decryptor = SymmetricKey.CreateDecryptor(KeyBytes,
InitialVectorBytes)){

using (MemoryStream MemStream = new MemoryStream(CipherTextBytes))
                    {
using (CryptoStream CryptoStream = new CryptoStream(MemStream, Decryptor,
CryptoStreamMode.Read))
                        {
ByteCount = CryptoStream.Read(PlainTextBytes, 0, PlainTextBytes.Length);
MemStream.Close();
CryptoStream.Close();
                    }
                }
        }
        SymmetricKey.Clear();
        return Encoding.UTF8.GetString(PlainTextBytes, 0, ByteCount);
```

7.2 AES v Javě

Stejně jako u šifry TripleDES si i tady vystačíme s třídou Cipher. Inicializujeme ji, definujeme mod CBC a výplň PKCS5. Oproti TripleDES si ale ukážeme, jak si klíč vygenerovat jednoduše pomocí třídy KeyGenerator. Výhoda je, že se nemusíme starat o klíč a inicializační vektor.

Zdrojový kód:

```
// vytvoření instance KeyGeneratoru, definování šifry jako AES
KeyGenerator keygenerator = KeyGenerator.getInstance("AES");

// tvorba klíče pomocí generování
SecretKey secKey = keygenerator.generateKey();

// vytvoření instance šifry
Cipher aesCipher;

// inicializace šifry s definováním typu, modu a výplně
aesCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

// inicializace pro šifrování
aesCipher.init(Cipher.ENCRYPT_MODE, secKey);

// důvěrné informace které chceme zašifrovat
byte[] text = "Test of AES cipher".getBytes();

System.out.println("Text [Byte Format] : " + text);
System.out.println("Text : " + new String(text));
```

```
// zašifrování důvěrných informací
byte[] textEncrypted = aesCipher.doFinal(text);

System.out.println("Text Encrypted : " + textEncrypted);

// inicializace té samé šifry pro dešifrování, získání IV
AlgorithmParameters params = aesCipher.getParameters();
byte[] iv = params.getParameterSpec(IvParameterSpec.class).getIV();
aesCipher.init(Cipher.DECRYPT_MODE, secKey, new IvParameterSpec(iv));

// dešifrování zašifrované zprávy
byte[] textDecrypted = aesCipher.doFinal(textEncrypted);

System.out.println("Text Decrypted : " + new String(textDecrypted));
```

7.3 AES v PHP

U AES šifry v PHP využijeme knihovnu stejnou jako u TripleDES, a to Mcrypt. Algoritmus použijeme Rijndael pro 256 bitový klíč s modem OFB bez výplně, protože ji výplň Mcrypt nepodporuje. Klíč si vytvoříme stejným způsobem jako u TripleDES – hašovací funkcí.

Zdrojový kód:

```
// otevření modulu s definováním algoritmu a modu
$td = mcrypt_module_open('rijndael-256', '', 'ofb', '');

// vygenerování inicializačního vektoru
$iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_DEV_RANDOM);

// získání velikosti klíče a hašování textu pro vytvoření klíče
$ks = mcrypt_enc_get_key_size($td);
$key = substr(md5('very secret key'), 0, $ks);

// inicializace s předáním parametrů modulu, klíče a IV
mcrypt_generic_init($td, $key, $iv);

// původní text
$originalText = 'This is very important data';

// šifrování původního textu
$encrypted = mcrypt_generic($td, $originalText);

// deinicializace
mcrypt_generic_deinit($td);

// znovu inicializování s předáním paramterů modulu, klíče a IV
mcrypt_generic_init($td, $key, $iv);

// dešifrování šifrovaného textu
$decrypted = mdecrypt_generic($td, $encrypted);

// deinicializace a zavření modulu
mcrypt_generic_deinit($td);
mcrypt_module_close($td);

// výpis dat na obrazovku
echo "Original text: ".$originalText;
echo "\n Encrypted: ".$encrypted;
echo "\n Decrypted: ".$decrypted;
```

7.4 AES v Pythonu

Zde, stejně jako u TripleDES, máme na výběr z více variant. Můžeme použít M2Crypto, PyCrypto nebo Chillkat. Hlavní rozdíl mezi PyCrypto a M2Crypto je v absenci výplně u knihovny PyCrypto. Navíc M2Crypto je postaveno na OpenSSL, což poskytuje všechny funkce které bychom při šifrování mohli potřebovat. Další roli by mohly hrát i jiné faktory, jako verze Pythonu a OpenSSL a především prostředí v jakém chceme program provozovat. Například Google App nepodporujeM2Crypto. My si ukážeme PyCrypto, výplň zatím řešit nemusíme.

Zdrojový kód:

```
# importování AES z knihovny
from Crypto.Cipher import AES

# nastavení klíče, textové zprávy a inicialičního vektoru
pwd='abcdefghijklmnop'
txt = 'ea523a664dabaa4476d31226a1e3bab0'
Initial16bytes='0123456789ABCDEF'

# založení instance s klíčem, modem a IV
crypt = AES.new(pwd, AES.MODE_CBC,Initial16bytes)

# zašifrování původní textové zprávy
c = crypt.encrypt(txt)

# výpis původní zprávy a zašifrovaného textu
print "Original data: %r" % txt
print "Encrypted: %r" % c

# reinicializace
crypt = AES.new(pwd, AES.MODE_CBC,Initial16bytes)

# dešifrování zprávy
txt_plain=crypt.decrypt(c)

# výpis dešifrované zprávy
print "Encrypted: %r" % txt_plain
input()
```

8 RSA

V praxi se šifra RSA využívá hlavně pro dva účely – šifrování klíčů symetrických šifer a digitálnímu podpisu. Identifikace na základě použití RSA (např. SecureID) je dnes velice rozšířené, ať už se jedná o tokeny, karty nebo software.

U všech programů budeme používat ten samý klíč. Klíč jsme si vygenerovali pomocí OpenSSL a máme jeho dvě verze, ve formátu pem, což je zapsání ve formě ascii a ve formátu der, což je binární zápis. Je to z toho důvodu, že Java umí pracovat s der i bez použití externích knihoven. OpenSSL umí převádět z jednoho formátu do druhého a tak použití dvou formátu nepředstavuje žádný problém. U pem najdeme privátní klíč mezi řádky:

```
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
```

U veřejného klíče je místo „private“ napsáno „public“. Pokud známe privátní klíč tak si můžeme odvodit i veřejný klíč.

Tímto příkazem si v OpenSSL vygenereujeme soukromý klíč o velikosti 1024 bitů.

```
openssl genrsa -out private.pem 1024
```

Soukromý klíč obsahuje i veřejný klíč, použijeme tedy tento příkaz na jeho uložení.

```
openssl rsa -in private.pem -out public.pem -outform PEM -pubout
```

V posledním kroku si vytvoříme i klíče ve formátu der.

```
rsa -in private.key -inform PEM -out private.key -outform DER
```

8.1 RSA v C#

Zde musíme použít externí knihovnu BouncyCastle, abychom mohli načíst klíč ze souboru. Tato knihovna je poměrně rozšířená a poskytuje mnoho funkcí. My budeme potřebovat hlavně PemReader. Pro samotnou práci se šifrou RSA použijeme již integrovanou třídu RSACryptoServiceProvider.

Zdrojový kód:

```
// načtení klíče do proměnné
StreamReader sr = new
StreamReader("..\\..\\..\\..\\RSA\\Keys\\private_key.pem");

// založení PemReaderu pro čtení klíče
PemReader pr = new PemReader(sr);

// založení asymetrického klíče a zapsání klíče z PemReaderu
AsymmetricCipherKeyPair KeyPair = (AsymmetricCipherKeyPair)pr.ReadObject();

// proměnná pro parametry a určení privátního klíče
RSAParameters rsap = DotNetUtilities.ToRSAParameters
((RsaPrivateCrtKeyParameters)KeyPair.Private);

// instance, která nám bude zajišťovat šifrování
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

// importování parametrů
rsa.ImportParameters(rsap);

// původní zpráva
String origText = "Testing of RSA cipher";
```

```
// převod původní zprávy na pole bytů
byte[] plainbytes = System.Text.Encoding.UTF8.GetBytes(origText);

// zašifrování a převedení zašifrované zprávy do formátu Base64
byte[] cipherbytes = rsa.Encrypt(plainbytes, false);
String result = Convert.ToBase64String(cipherbytes);

// převedení zpět z Base64
byte[] getpassword = Convert.FromBase64String(result);

// dešifrování a převedení do řetězce
byte[] plain = rsa.Decrypt(getpassword, false);
String decryptedResult = System.Text.Encoding.UTF8.GetString(plain);
```

8.2 RSA v Javě

Jak už jsme si uvedli, v Javě budeme pracovat s klíčem ve formátu der. Jako u předchozích programů, i tady použijeme třídu Cipher. Výplň použijeme PKCS1 a mod ECB. Pro čtení klíče použijeme 2 různé metody.

Zdrojový kód:

Metoda pro čtení veřejného klíče

```
public static PublicKey get(String filename) throws Exception {

// načtení klíče ze souboru a uložení do pole bytů keyBytes
File f = new File(filename);
FileInputStream fis = new FileInputStream(f);
DataInputStream dis = new DataInputStream(fis);
byte[] keyBytes = new byte[(int)f.length()];
dis.readFully(keyBytes);
dis.close();

// tato třída reprezentuje práci s veřejným klíčem podle standardu ASN.1
X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);

// instance třídy KeyFactory pro práci s klíčema
KeyFactory kf = KeyFactory.getInstance("RSA");

// vracení klíče generovaného KeyFactory s parametrem podle načteného klíče
// ze souboru
return kf.generatePublic(spec);
}
```

Metoda pro čtení privátního klíče

```
public static PrivateKey get(String filename) throws Exception {

// načtení klíče ze souboru a uložení do pole bytů keyBytes
File f = new File(filename);
FileInputStream fis = new FileInputStream(f);
DataInputStream dis = new DataInputStream(fis);
byte[] keyBytes = new byte[(int)f.length()];
dis.readFully(keyBytes);
dis.close();

// tato třída reprezentuje práci s privátním klíčem podle standardu ASN.1
PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);

// instance třídy KeyFactory pro práci s klíčema
KeyFactory kf = KeyFactory.getInstance("RSA");
```

```
// vracení klíče generovaného KeyFactory s parametrem podle načteného klíče
// ze souboru
return kf.generatePrivate(spec);
}
```

Šifrování

```
// instance třídy Cipher s modem ECB a výplní PKCS1
Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

// inicializace pro šifrování s použitím veřejného klíče
cipher.init(Cipher.ENCRYPT_MODE, publicKey);

// zašifrování dat a uložení do pole bytů
byte[] encrypted = cipher.doFinal(data);
```

Dešifrování

```
// instance třídy Cipher s modem ECB a výplní PKCS1
Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

// inicializace pro dešifrování s použitím veřejného klíče
cipher.init(Cipher.DECRYPT_MODE, privateKey);

// dešifrování dat a uložení do pole bytů
byte[] decrypted = cipher.doFinal(encrypetd);

// převedení na řetězec
String result = new String(decrypted);
```

Tělo programu

```
// instance třídy PublicKey do které se uloží výsledek metody pro čtení
// veřejného klíče
PublicKey publicKey = PublicKeyReader.get("..\\RSA\\Keys\\public_key.der");

// instance třídy PrivateKey do které se uloží výsledek metody pro čtení
// privátního  klíče
PrivateKey privateKey =
PrivateKeyReader.get("..\\RSA\\Keys\\private_key.der");

// originální data která chceme zašifrovat
String data = "Test of RSA cipher";

// volání metod pro šifrování a dešifrování, výpis dat
byte[] encrypted = encode(publicKey, data.getBytes());
String decrypted = new String(decode(privateKey, encrypted));
System.out.println(data);
System.out.println(encrypted);
System.out.println(decrypted);
```

8.3 RSA v PHP

U PHP použijeme přímo funkce knihovny openssl, jelikož jsme si samotný klíč vygenerovali pomocí OpenSSL, tak s ním budeme i pomocí této knihovny pracovat.

Zdrojový kód:

```
// originální data určené k šifrování
$string = "data to encrypt";
```

```
// získání veřejného klíče ze souboru pomocí funkce knihovny OpenSSL
$publickey = openssl_pkey_get_public
(file_get_contents("..\Keys\public_key.pem"));

// získání soukromého klíče ze souboru
$privatekey = openssl_pkey_get_private
(file_get_contents("..\Keys\private_key.pem"));

// zašifrování veřejným klíčem
openssl_public_encrypt($string, $crypt_output, $publickey);

// dešifrování soukromým klíčem
openssl_private_decrypt($crypt_output, $decrypted, $privatekey);

// výpis dat na obrazovku
echo "Original text: ".$string;
echo "\n\nEncrypted data: ".$crypt_output;
echo "\n\nDecrypted data: ".$decrypted;
```

8.4 RSA v Pythonu

U Pythonu máme na výběr jako vždy z více možností. Jako u předchozích šifer, i tady můžeme použít Chillkat a M2Crypto, obojí knihovny podporují výplň podle standardu PKCS#1. Další možností je použít Python-RSA knihovnu, která podporuje výplň PKCS#1, generování klíčů i podepisování a ověřování. Jedná se o knihovnu čistě pro práci s RSA. V neposlední řade je tu také PyCrypto, kterou si nyní ukážeme.

Zdrojový kód:

```
# importování z knihovny PyCrypto
from Crypto.PublicKey import RSA
from Crypto.Util import asn1

# otevření veřejného klíče, přečtení a vložení do proměnné
f = open("../RSA/Keys/public_key.pem")
public_key = RSA.importKey(f.read())

# původní zpráva
orig_text = "Testing of RSA cipher"

# zašifrování původní zprávy a uložení do souboru
enc_data = public_key.encrypt(orig_text, 32)

# otevření soukromého  klíče, přečtení a vložení do proměnné
fp = open("../RSA/Keys/private_key.pem")
private_key = RSA.importKey(fp.read())

# dešifrování a uložení do proměnné
dec_data = private_key.decrypt(enc_data)

# tisk dat na obrazovku
print "Original text: %r" % orig_text
print "Encrypted: %r" % enc_data
print "Decrypted: %r" % dec_data
input()
```

9 Kooperace jazyků

V této třetí a poslední části si ukážeme jak mohou jednotlivé jazyky spolupracovat. U každé šifry si popíšeme šifrovací a dešifrovací algoritmus a řekneme si, co musíme splnit abychom byli schopni šifrovat v jednom jazyce a dešifrovat v jiném. Všechny soubory pro klíče, původní zprávy, zašifrované zprávy a popřípadě i inicializační vektory ukládáme na disk tak, aby k nim měly přístup všechny programy. U každého programovacího jazyku budeme mít dva samostatné programy, jeden pro šiforvání (encrypt) a druhý pro dešifrování (decrypt).

9.1 Triple DES

Klíč používáme v délce 24 bytů, IV v dílce 8 bytů. Zašifrovanou zprávu kódujeme Base64 a ukládáme do souboru. Klíč, IV i původní zpráva jsou zapsány v samostatných textových souborech bez dalších úprav. Ve všech jazycích používáme mod CBC, výplň PKCS#7 a kódování a dekódování Base64. Při šifrování tedy načítáme klíč, IV a původní zprávu, šifrovaný text následně kódujeme Base64 a ukládáme do souboru. Při dešifrování nejprve dekódujeme z Base64 a následně dešifrujeme, přičemž musíme mít již načtenou šifrovanou zprávu, klíč a IV.

Šifrování	Dešifrování			
	C#	Java	PHP	Python
C#	--	ECB, CBC PKCS#7	ECB, CBC PKCS#7	ECB, CBC PKCS#7
Java	ECB, CBC PKCS#5	--	ECB, CBC PKCS#5	ECB, CBC PKCS#5
PHP	ECB, CBC PKCS#7	ECB, CBC PKCS#7	--	ECB, CBC PKCS#7
Python	ECB, CBC PKCS#5	ECB, CBC PKCS#5	ECB, CBC PKCS#5	--

Srovnání jazyků u Triple DES

9.1.1 Triple DES v C#

Zde je kód téměř totožný s naším příkladem ve druhé části této práce. Používáme zde několik streamů a pro šifrování TriplEDSCryptoServiceProvider

Šifrování

```
// metoda pro šifrování s parametry pro hodnotu, klíč a IV
public static string EncryptTripleDES(string value, string key, string iv)
        {
byte[] encryptionKey = Encoding.UTF8.GetBytes(key);
byte[] initializationVector = Encoding.UTF8.GetBytes(iv);

// inicializace provideru
TripleDESCryptoServiceProvider cryptoProvider = new
TripleDESCryptoServiceProvider();
```

```
MemoryStream ms = new MemoryStream();

// vytvoření encryptoru
CryptoStream cs = new CryptoStream(ms,
cryptoProvider.CreateEncryptor(encryptionKey, initializationVector),
CryptoStreamMode.Write);
StreamWriter sw = new StreamWriter(cs);
sw.Write(value);
sw.Flush();
cs.FlushFinalBlock();
ms.Flush();

// na konci musíme data převést a zakódovat do Base64
            return Convert.ToBase64String(ms.GetBuffer(), 0, (int)ms.Length);
}
```

Dešifrování

```
// metoda pro dešifrování s parametry pro šifrovaný vstup, klíč a IV
public static String DecryptTripleDES(String input, string key, string iv)
        {
// nejprve musíme dekódovat z Base64
byte[] inputArray = Convert.FromBase64String(input);

// inicializace provideru
TripleDESCryptoServiceProvider tripleDES = new
TripleDESCryptoServiceProvider();
tripleDES.Key = UTF8Encoding.UTF8.GetBytes(key);
tripleDES.IV = UTF8Encoding.UTF8.GetBytes(iv);

// vytvoření decryptoru
ICryptoTransform transform = tripleDES.CreateDecryptor();
byte[] resultArray = transform.TransformFinalBlock(inputArray, 0,
inputArray.Length);
tripleDES.Clear();
return UTF8Encoding.UTF8.GetString(resultArray);
}
```

9.1.2 Triple DES v Javě

V Javě budeme používat standard výplně PKCS#5, protože je prakticky identický se standardem PKCS#7. Jako v minulých případech, i tady použijeme třídu Cipher.

Šifrování

```
// metoda pro šifrování s parametry text, klíč a IV
public static String EncryptString(String text, String key, String iv)
throws Exception {

// inicializace třídy
Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");

// definování klíče na velikost 24 bytů a naplnění pole, kontrola velikosti a
// kopírování do nové proměnné
byte[] keyBytes= new byte[24];
byte[] b= key.getBytes("UTF-8");
int len= b.length;
if (len > keyBytes.length) len = keyBytes.length;
System.arraycopy(b, 0, keyBytes, 0, len);

// specifikace pro klíč a inicializační vektor
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "DESede");
IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

// inicializování šifry pro šifrování
cipher.init(Cipher.ENCRYPT_MODE,keySpec,ivSpec);
```

```
// zašifrování textu
byte[] results = cipher.doFinal(text.getBytes("UTF-8"));

// převedení do Base64 a navrácení výsledku
BASE64Encoder encoder = new BASE64Encoder();
return encoder.encode(results);
}
```

Dešifrování

```
// metoda pro dešifrování s parametry text, klíč a IV
public static String DecryptString(String text, String key, String iv) throws
Exception
{

// inicializace třídy Cipher
Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");

// definování klíče na velikost 24 bytů a naplnění pole, kontrola velikosti a
// kopírování do nové proměnné
byte[] keyBytes= new byte[24];
byte[] b= key.getBytes("UTF-8");
int len= b.length;
if (len > keyBytes.length) len = keyBytes.length;
System.arraycopy(b, 0, keyBytes, 0, len);

// specifikace pro klíč a inicializační vektor
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "DESede");
IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

// dešifrování
cipher.init(Cipher.DECRYPT_MODE,keySpec,ivSpec);

BASE64Decoder decoder = new BASE64Decoder();
byte [] results = cipher.doFinal(decoder.decodeBuffer(text));
return new String(results,"UTF-8");
}
```

9.1.3 Triple DES v PHP

U PHP je třeba vyřešit problém s výplní. Zvolíme tedy skvělou externí knihovnu autora Matthew Dahla pro šifrování pomocí Mcrypt. Původně je tato knihovna pro šifru AES, ovšem s malými úpravami ji můžeme použít i zde. V knihovně je vyřešeno vše co bychom mohli potřebovat, nás hlavně zajímá mod a výplň. Podporované mody jsou všechny a výplň je zde defaultně PKCS#7, což potřebujeme. Práce s touto knihovnou je stejně jednoduchá jako práce s jinými knihovnami.

Šifrování

```
// import knihovny
require_once('3DESCipher.class.php');

// inicializace třídy s parametrem modu, výplň je automaticky nastavena na
// PKCS#7
$tripledes = new TRIPLEDESCipher(MCRYPT_MODE_CBC);

// otevření souboru pro zapisování do kterého uložíme šifrovaný text
$file = fopen('..\3DES\encrypted.txt', "w+");

// zašifrování originálního textu
$encrypted = $tripledes->encrypt($orig, $key, $iv);

// zakódování do Base64, zapsání textu do souboru a uzavření souboru
fwrite($file, base64_encode($encrypted));
```

```
fclose($file);
```

Dešifrování

```
// import knihovny
require_once('3DESCipher.class.php');

// inicializace třídy
$tripledes = new TRIPLEDESCipher(MCRYPT_MODE_CBC);

// dekódování z Base64  a dešifrování
$enc_data = base64_decode($tmp);
$decrypted = $tripledes->decrypt($enc_data, $key, $iv);

// výpis dat na obrazovku
echo "\n\nDecrypted data: ".$decrypted;
```

9.1.4 Triple DES v Pythonu

Zde použijeme knihovnu PyDES, hlavním důvodem je již integrovaná výplň PKCS#5. To je nutnost, abychom byli schopni dešifrovat z jiných jazyků a naopak, aby programy psané v jiných jazycích byly schopné dešifrovat text zašifrovaný v Pythonu.

Šifrování

```
# inicializace šifry s klíčem, modem, inicializačním vektorem a výplní
cipher = triple_des(key, CBC, iv, pad=None, padmode=PAD_PKCS5)

# zašifrování textu
encrypted = cipher.encrypt(orig_text)

# zakódování šifrovaného textu Base64
enc_cipher = base64.b64encode(encrypted)

# uložení výsledného textu do souboru
encryptedFile=file("../3DES/encrypted.txt", 'w')
encryptedFile.write(enc_cipher)
encryptedFile.close()
```

Dešifrování

```
# načtení šifrované zprávy do proměnné tmp
# inicializace šifry s klíčem, modem, IV a výplní
cipher = triple_des(key, CBC, iv, pad=None, padmode=PAD_PKCS5)

# dekódování z Base64 a následné dešifrování zprávy
encrypted = base64.b64decode(tmp)
decrypted = cipher.decrypt(encrypted)
```

9.2 AES

Používáme zde klíč o délce 16 bytů, stejnou délku má i inicializační vektor. Abychom mohli úspěšně šifrovat a dešifrovat v odlišných jazycích, tak budeme používat výplň PKCS#7, popř. PKCS#5 která je téměř identická. Při šifrování načítáme klíč, IV a původní zprávu, šifrovaný text následně kódujeme Base64 a ukládáme do souboru. Při dešifrování nejprve dekódujeme z Base64 a následně dešifrujeme, přičemž musíme mít již načtenou šifrovanou zprávu, klíč a IV.

Šifrování	Dešifrování			
	C#	Java	PHP	Python
C#	--	ECB, CBC PKCS#7	ECB, CBC PKCS#7	ECB, CBC, CFB PKCS#7
Java	ECB, CBC PKCS#5	--	ECB, CBC PKCS#5	ECB, CBC PKCS#5
PHP	ECB, CBC PKCS#7	ECB, CBC PKCS#7	--	ECB, CBC PKCS#7
Python	ECB, CBC, CFB PKCS#5	ECB, CBC PKCS#5	ECB, CBC PKCS#5	--

Srovnání jazyků u AES

9.2.1 AES v C#

Opět budeme používat RijndaelManaged pro práci s algoritmem.

Šifrování

```
// metoda pro šifrování, s parametry původní text, klíč a IV
public static string Encrypt(string textToEncrypt, string key, string iv)
{
// inicializace třídy pro šifrování
RijndaelManaged rijndaelCipher = new RijndaelManaged();

// převedení klíče na formát byte a kontrola délky klíče
byte[] pwdBytes = Encoding.UTF8.GetBytes(key);
byte[] keyBytes = new byte[0x10];
int len = pwdBytes.Length;
if (len > keyBytes.Length)
{
       len = keyBytes.Length;
}
Array.Copy(pwdBytes, keyBytes, len);

// specifikace klíče který bude použit
rijndaelCipher.Key = keyBytes;

// převedení a kontrola IV
byte[] ivBytes = Encoding.UTF8.GetBytes(iv);
byte[] iv2Bytes = new byte[0x10];
int len2 = ivBytes.Length;
if (len2 > iv2Bytes.Length)
{
       len2 = iv2Bytes.Length;
}
Array.Copy(ivBytes, iv2Bytes, len2);

// specifikace IV
rijndaelCipher.IV = iv2Bytes;

// vytvoření encryptoru
ICryptoTransform transform = rijndaelCipher.CreateEncryptor();

// šifrování a převedení do Base64
byte[] plainText = Encoding.UTF8.GetBytes(textToEncrypt);
```

```
return Convert.ToBase64String(transform.TransformFinalBlock(plainText, 0,
plainText.Length));
}
```

9.2.2 AES v Javě

I zde budeme používat třídu Cipher s modem CBC a výplní PKCS5.

Šifrování

```
public static String Encrypt(String text, String key, String iv)
      throws Exception {

// vytvoření instance
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

// převedení klíče na formát byte a kontrola délky
byte[] keyBytes= new byte[16];
byte[] b= key.getBytes("UTF-8");
int len= b.length;
if (len > keyBytes.length) len = keyBytes.length;
System.arraycopy(b, 0, keyBytes, 0, len);

// specifikace klíče a inicializačního vektoru
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

// inicializace šifry pro šifrování
cipher.init(Cipher.ENCRYPT_MODE,keySpec,ivSpec);

// šifrování a kódování do Base64
byte[] results = cipher.doFinal(text.getBytes("UTF-8"));
BASE64Encoder encoder = new BASE64Encoder();
return encoder.encode(results);
}
```

Dešifrování

```
public static String Decrypt(String text, String key, String iv) throws
Exception{
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
byte[] keyBytes= new byte[16];
byte[] b= key.getBytes("UTF-8");
int len= b.length;
if (len > keyBytes.length) len = keyBytes.length;
System.arraycopy(b, 0, keyBytes, 0, len);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

// inicializace pro dešifrování
cipher.init(Cipher.DECRYPT_MODE,keySpec,ivSpec);

// dekódování z Base64
BASE64Decoder decoder = new BASE64Decoder();

// dešifrování
byte [] results = cipher.doFinal(decoder.decodeBuffer(text));
return new String(results,"UTF-8");
}
```

9.2.3 AES v PHP

Pro šifrování v PHP použijeme původní třídu, jejíž přepsanou verzi jsme použili u Triple DES.

Šifrování

```
// načtení knihovny
require_once('AESCipher.class.php');

// vytvoření instance třídy
$aes = new AESCipher(MCRYPT_MODE_CBC);

// šifrování a ukládání do souboru v Base64
$file = fopen('..\AES\encrypted.txt', "w+");
$encrypted = $aes->encrypt($orig, $key, $iv);
fwrite($file, base64_encode($encrypted));
fclose($file);
```

Dešifrování

```
// načtení knihovny
require_once('AESCipher.class.php');

// vytvoření instance třídy
$aes = new AESCipher(MCRYPT_MODE_CBC);

// dekódování dat ze souboru
$enc_data = base64_decode($tmp);

// dešifrování
$decrypted = $aes->decrypt($enc_data, $key, $iv);
```

9.2.4 AES v Pythonu

Zde si ukážeme použití knihoven PyCrypto a PKCS7, která zajišťuje výplň podle PKCS#7. Lze ji použít i pro jiné šifry, u kterých bychom tuto výplň chtěli.

Šifrování

```
# import knihoven
from Crypto.Cipher import AES
import base64
from pkcs7 import PKCS7Encoder

# definování modu
mode = AES.MODE_CBC

# deklarace kodéru
encoder = PKCS7Encoder()

# vytvoření instance třídy AES
encryptor = AES.new(key, mode, iv)

# přidání výplně k textu, zašifrování a kódování a do Base64
pad_text = encoder.encode(text)
cipher = encryptor.encrypt(pad_text)
enc_cipher = base64.b64encode(cipher)
```

Dešifrování

```
# import knihoven
from Crypto.Cipher import AES
import base64
from pkcs7 import PKCS7Encoder

# definování modu
mode = AES.MODE_CBC
```

```
# deklarace kodéru, v tomto případě se jedná o dekódování
decoder = PKCS7Encoder()

# vytvoření instance třídy AES
decryptor = AES.new(key, mode, iv)

# dekódování z Base64, dešifrování a odstranění výplně
text = base64.b64decode(tmp)
decrypted_data = decryptor.decrypt(text)
dec_data = decoder.decode(decrypted_data)
```

9.3 RSA

Zde použijeme úplně stejné klíče jako ve druhé části. Jedná se o veřejný a soukromý klíč, veřejný je určený k šifrování a soukromý k dešifrování. V každém programu je tedy potřeba načíst veřejný klíč a původní zprávu při šifrování a soukromý klíč a zašifrovanou zprávu při dešifrování. Zároveň zde používáme výplň PKCS#1, která je určena pro šifru RSA.

Šifrování			Dešifrování	
	C#	Java	PHP	Python
C#	--	PKCS#1, OAEP	PKCS#1, OAEP	PKCS#1
Java	PKCS#1, OAEP	--	PKCS#1, OAEP	PKCS#1
PHP	PKCS#1, OAEP	PKCS#1, OAEP	--	PKCS#1
Python	PKCS#1	PKCS#1	PKCS#1	--

Srovnání jazyků u RSA

9.3.1 RSA v C#

V C# je trochu změna oproti jiným jazykům, při šifrování i dešifrování používáme jen soukromý klíč. Pokud bychom chtěli šifru použít, pak samozřejmě musíme používat i veřejný klíč, ale pro účely prozkoumání spolupráce mezi jazyky nám to postačí takto. Z privátního klíče si totiž umíme vytáhnout i klíč veřejný.

Šifrování

```
// stream pro čtení s relativní adresou soukromého klíče
StreamReader sr = new
StreamReader("..\\..\\..\\..\\RSA\\Keys\\private_key.pem");

// třída PemReader BouncyCastle pro čtení klíče ve formátu .pem
PemReader pr = new PemReader(sr);

// asymetrický pár klíčů do kterého se uloží objekt z PemReaderu
AsymmetricCipherKeyPair KeyPair = (AsymmetricCipherKeyPair)pr.ReadObject();
```

```
// deklarace parametrů
RSAParameters rsap =
DotNetUtilities.ToRSAParameters((RsaPrivateCrtKeyParameters)KeyPair.Private);

// nová instance pro práci s RSA
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

// importování parametrů
rsa.ImportParameters(rsap);

String origText = getString("..\\..\\..\\..\\RSA\\original.txt");
byte[] plainbytes = System.Text.Encoding.UTF8.GetBytes(origText);

// šifrování a převedení do Base64
byte[] cipherbytes = rsa.Encrypt(plainbytes, false);
String data = Convert.ToBase64String(cipherbytes);
```

Dešifrování

```
StreamReader sr = new StreamReader("D:\\RSA\\Keys\\private_key.pem");
PemReader pr = new PemReader(sr);
AsymmetricCipherKeyPair KeyPair = (AsymmetricCipherKeyPair)pr.ReadObject();
RSAParameters rsap =
DotNetUtilities.ToRSAParameters((RsaPrivateCrtKeyParameters)KeyPair.Private);
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
rsa.ImportParameters(rsap);
String origText = getString("..\\..\\..\\..\\RSA\\original.txt");
byte[] cipherbytes =
Convert.FromBase64String(getString("..\\..\\..\\..\\RSA\\encrypted.txt"));

// dešifrování
byte[] plain = rsa.Decrypt(cipherbytes, false);
String decryptedResult = System.Text.Encoding.UTF8.GetString(plain);
```

9.3.2 RSA v Javě

Zde využijeme třídu Cipher s parametry RSA, modem ECB a výplní PKCS#1. Veškeré metody pro získání veřejného a soukromého klíče byly již popsány ve druhé části.

Šifrování

```
public static byte[] encode(PublicKey publicKey, byte[] data)
                    throws NoSuchAlgorithmException, NoSuchPaddingException,
                    InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {

// vytvoření instance třídy s názve šifry, modem a výplní
Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

// inicializace šifry pro šifrování
cipher.init(Cipher.ENCRYPT_MODE, publicKey);

// šifrování
byte[] encrypted = cipher.doFinal(data);
return encrypted;
}
```

Dešifrování

```
public static String decode(PrivateKey privateKey, byte[] encrypetd)
                    throws NoSuchAlgorithmException, NoSuchPaddingException,
```

```
                    InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {

// vytvoření instance
Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

// inicializování pro dešifrování
cipher.init(Cipher.DECRYPT_MODE, privateKey);

// dešifrování
byte[] decrypted = cipher.doFinal(encrypetd);
String result = new String(decrypted);
return result;
}
```

9.3.3 RSA v PHP

Pro práci s RSA použijeme OpenSSL, které nám umožní načítat klíče i samotné šifrování. Ukládání do proměnné probíhá rovnou v Base64, takže nemusíme výsledek zvlášť kódovat.

Šifrování

```
// načtení veřejného klíče
$publickey = openssl_pkey_get_public
(file_get_contents("..\RSA\Keys\public_key.pem"));

// otevření souboru pro zapisování
$file = fopen('..\RSA\encrypted.txt', "w");

// zašifrování originální zprávy a uložení do proměnné crypt_output
openssl_public_encrypt($orig, $crypt_output, $publickey);
fwrite($file, base64_encode($crypt_output));
fclose($file);
```

Dešifrování

```
// načtení soukromého klíče
$privatekey =
openssl_pkey_get_private(file_get_contents("..\RSA\Keys\private_key.pem"));

// dešifrování
openssl_private_decrypt(base64_decode($tmp), $decrypted, $privatekey);
```

9.3.4 RSA v Pythonu

Nabízí se použít knihovnu Python-RSA, která by nám zajistila načítání klíčů ve formátu pem nebo der i šifrování s PKCS#1. Další variantou je M2Crypto.

Šifrování

```
# import knihoven
import base64
from base64 import b64encode
from base64 import b64decode
import M2Crypto
from M2Crypto import RSA,SSL

# načtení veřejného klíče
public_key = M2Crypto.RSA.load_pub_key("../RSA/Keys/public_key.pem")

# šifrování
encrypted = public_key.public_encrypt(orig_text, M2Crypto.RSA.pkcs1_padding)
```

Dešifrování

```
# import knihoven
import base64
from base64 import b64encode
from base64 import b64decode
import M2Crypto
from M2Crypto import RSA,SSL

# načtení soukromého klíče
private_key = M2Crypto.RSA.load_key("../RSA/Keys/private_key.pem")

# dekódování z Base64
encrypted_data = base64.b64decode(tmp)

# dešifrování
dec_data = private_key.private_decrypt(encrypted_data,
M2Crypto.RSA.pkcs1_padding)
```

10 Závěr

V této práci jsme se zabývali šifrovacími algoritmy a jejich implementací v programovacích jazycích C#, Java, PHP a Python a spoluprací mezi těmito jazyky. V první části jsme se dozvěděli o kryptografii obecně, jak dělíme šifry a jaké mohou mít parametry. Dále jsme se dočetli o jednotlivých šifrách, jejich slabinách a silných stránkách a k čemu se v praxi dají využít. Ve druhé části se čtenář dozvídá jak tyto šifry implementovat v jednotlivých jazycích, kdy si vystačí s integrovanými knihovnami a kdy musí sáhnout po externích zdrojích. Ve třetí části se nalézá doporučení jak skloubit tyto jazyky dohromady pro případnou spolupráci a co všechno se musí dodržet, aby taková spolupráce fungovala. Existuje spousta uživatelů, kteří řeší problém spolupráce dvou programovacích jazyků u jedné šifry a není moc publikací nebo zdrojů, které by tento problém řešily pohromadě. Z toho důvodu by mohla být tato práce přínosem na poli kooperace programovacích jazyků.

Použitá literatura

[1] LITZENBERGER, Dwayne. *PyCrypto* [online]. [cit. 2013-05-06]. Dostupné z: https://www.dlitz.net/software/pycrypto/

[2] M2Crypto 0.21.1. PYTHON SOFTWARE FOUNDATION. M2Crypto 0.21.1 [online]. 1990 [cit. 2013-05-06]. Dostupné z: https://pypi.python.org/pypi/M2Crypto

[3] Cryptography Extensions. *PHP Manual* [online]. 2001, 3.5.2013 [cit. 2013-05-06]. Dostupné z: http://php.net/manual/en/refs.crypto.php

[4] PyDes. In: *Whitemans Blog* [online]. 2010 [cit. 2013-05-06]. Dostupné z: http://twhiteman.netfirms.com/des.html

[5] Základy šifrování v Javě. In: ŠINDELÁŘ, Pavel. *Vsaď na Javu* [online]. 2011, 2.5.2011 [cit. 2013-05-06]. Dostupné z: http://vsadnajavu.cz/2011-05/java-j2ee/zaklady-sifrovani-v-jave/

[6] RSA module for Python. DAHL, Matthew. *Blog - Stüvel photography* [online]. 2012 [cit. 2013-05-06]. Dostupné z: http://stuvel.eu/rsa

Seznam příloh

Součástí BP/DP je CD/DVD.

Bakalářská práce

Bakalářská práce.pdf

2. část

3. část